

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2001-306191

(P2001-306191A)

(43) 公開日 平成13年11月2日 (2001.11.2)

(51) Int.Cl.	識別記号	F I	テーマコード (参考)
G 0 6 F 1/26		H 0 2 J 1/00	3 0 4 D 5 B 0 1 1
H 0 2 J 1/00	3 0 4	G 0 6 F 1/00	3 3 4 H 5 G 0 6 5

審査請求 未請求 請求項の数10 O L (全 29 頁)

(21) 出願番号 特願2000-119033(P2000-119033)

(22) 出願日 平成12年4月20日 (2000.4.20)

(71) 出願人 000002185

ソニー株式会社

東京都品川区北品川6丁目7番35号

(72) 発明者 田島 茂

東京都品川区東五反田3丁目14番13号 株式会社ソニーコンピュータサイエンス研究所内

(74) 代理人 100101801

弁理士 山田 英治 (外2名)

Fターム (参考) 5B011 DA02 DA11 DA12 DA13 DB21

DB26 DC06 GC02 HH02 JB10

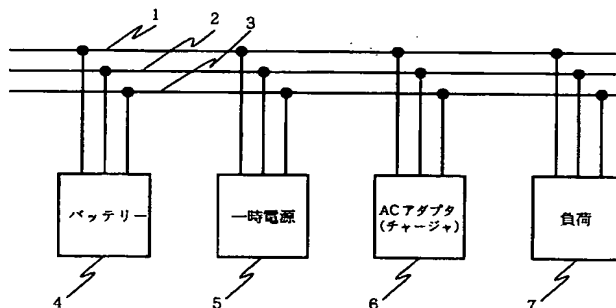
5G065 EA02 EA04 PA05

(54) 【発明の名称】 電源サーバ、電源クライアントおよび電源バスシステム

(57) 【要約】

【課題】 電源や負荷を統合的な環境で利用できるようにする。

【解決手段】 バッテリ4、一時電源5、ACアダプタ6、負荷7が電源バスライン1、接地バスライン2およびデータバスライン (バスシステム) に接続されている。バスシステムに接続されている各ブロックは自らをオブジェクトとして記述し、各ブロックのオブジェクトが、データバスラインを介して相互に状態データ等のやり取りを行う。各ブロックのオブジェクトは、他のブロックのオブジェクトからの要求に基づいて状態データを生成し、これを回答する。回答データを受け取ったブロックのオブジェクトはこれに基づいて電源の供給や消費を制御する。



## 【特許請求の範囲】

【請求項 1】 電源バス接続ポートと、

データベース接続ポートと、

当該データベース接続ポートより電源クライアントの状態を取得する手段と、

当該電源バス接続ポートを介して電力を供給する手段とを有することを特徴とする電源サーバ。

【請求項 2】 電源バス接続ポートと、

データベース接続ポートと、

当該データベース接続ポートに対して状態を送信する手段と、

当該電源バス接続ポートを介して電力を受け取る手段とを有することを特徴とする電源クライアント。

【請求項 3】 電源サーバと、

電源クライアントと、

電源サーバから電源クライアントに対して電源を供給する電源バスとを具備し、

当該電源クライアントは、

電源バス接続ポートと、

データベース接続ポートと、

当該データベース接続ポートに対して状態を送信する手段と、

当該電源バス接続ポートを介して電力を受け取る手段とを有し、

当該電源サーバは、

電源バス接続ポートと、

データベース接続ポートと、

当該データベース接続ポートより電源クライアントの状態を取得する手段と、

当該電源バス接続ポートを介して電力を供給する手段とを有することを特徴とする電源バスシステム。

【請求項 4】 電源バス接続ポートと、

データベース接続ポートと、

当該データベース接続ポートを伝送されるデータを受け取る手段と、

受け取った上記データに基づいて電力の消費を制御する手段とを有することを特徴とする電源消費ブロック。

【請求項 5】 電源バス接続ポートと、

データベース接続ポートと、

当該データベース接続ポートを伝送されるデータを受け取る手段と、

受け取った上記データに基づいて電力の供給を制御する手段とを有することを特徴とする電源供給ブロック。

【請求項 6】 少なくとも 1 つの電源供給ブロックと、

少なくとも 1 つの電源消費ブロックと、

電源バスおよびデータベースを具備し上記電源供給ブロックから上記電源消費に対して電源を供給するバスシステムとを有し、

上記電源消費ブロックは、

電源バス接続ポートと、

データベース接続ポートと、

上記データベース接続ポートを介してデータを送受信する手段と、

上記データベース接続ポートを介して受け取ったデータに基づいて上記電源バスを介して送られる電力の消費を制御する手段とを有し、

上記電源供給ブロックは、

電源バス接続ポートと、

データベース接続ポートと、

上記データベース接続ポートを介してデータを送受信する手段と、

上記データベース接続ポートを介して受け取ったデータに基づいて上記電源バスへの電源の供給を制御する手段とを有することを特徴とする電源システム。

【請求項 7】 上記電源消費ブロックおよび電源供給ブロックの属性がデータおよび関数として定義され、これらデータが上記データベースを介して上記電源供給ブロックおよび電源消費ブロックの間で送受信される請求項 6 記載の電源システム。

【請求項 7】 上記電源消費ブロックおよび電源供給ブロックの属性がデータおよび関数として定義され、これらデータが上記データベースを介して上記電源供給ブロックおよび電源消費ブロックの間で送受信される請求項 6 記載の電源システム。

【請求項 8】 上記電源消費ブロックおよび電源供給ブロックの少なくとも一部が上記バスシステムに対して着脱可能である請求項 6 または 7 記載の電源システム。

【請求項 9】 上記電源消費ブロックおよび電源供給ブロックの属性が、データとその操作アルゴリズム（関数）、継承、ポリモーフィズムを用いて記述される請求項 6、7 または 8 記載の電源システム。

【請求項 10】 上記バスシステムはデータベース管理の為にアドレス信号を周期的に発生し、そのアドレス信号に対応する情報は、上記バスシステムに接続されるブロックが発信する請求項 6、7、8 または 9 記載の電源システム。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】 この発明は、種々の電源や負荷を統合的に管理する技術に関する。

【0002】

【従来の技術】 バッテリー等を電源とする、ポータブル機器の発展にともない、複数のバッテリー搭載機器を同時に使用する機会が増加しつつある。例えば、モバイルコンピュータの画像を HMD（ヘッドマウンテッド・ディスプレイ）で観測する場合や、モバイルコンピュータをモデムカードを仲介として携帯電話に接続する場合などである。この様な場合においても、現在のところモバイル PC（パーソナル・コンピュータ）にはそれ専用の、また HMD や携帯電話にはそれら専用のバッテリーが用いられ、お互いの間でのエネルギーの融通は無い。

【0003】 また、PC やその周辺機器の発展に伴いウェアラブル PC が提唱され、いくつかの商品化例も見られるようになった。この場合でも、電源に関しては専用設計されたものが使用され、例えば手近に満充電された二

次電池があろうとも互換性が無いため流用できない。

【0004】一方、ウェアラブルPCを含むモバイル機器では、いかに電源を確保するかというのは切実な問題であり、場合によっては手回し発電機のような機器さえ必要になる。

【0005】しかしながら、現在では、各種電源や各種負荷（機器）の間で統一的に電源を制御するアーキテクチャは無く、二次電池やACアダプタ、簡易発電機等を設計するにしてもある特定の機器専用となってしまう。

【0006】

【発明が解決しようとする課題】本発明は、上記に述べた不具合を解決するため、電源供給源と電源消費者という考えに基づいて、それらを抽象化し、統一的に定義し、複数の電源供給源（バッテリーやACアダプタ等）、複数の電源消費者が一つの共通バスライン上に柔軟に着脱可能となるシステムを提供することを目的としている。

【0007】

【課題を解決するための手段】本発明によれば、上述の目的を達成するために、特許請求の範囲に記載のとおり10の構成を採用している。ここでは、特許請求の範囲の記載について補足的に説明をしておく。

【0008】すなわち、本発明によれば、電源消費ブロックや電源供給ブロックに、電源バス接続ポートと、データバス接続ポートと、当該データバス接続ポートを伝送されるデータを受け取る手段と、受け取った上記データに基づいて電力の消費や電源の供給を制御する手段とを設けている。

【0009】電源消費ブロックの中には、電源供給をあわせて行えるものがあってもよい。電源供給ブロックの中に電源消費をあわせて行えるものがあってもよい。

【0010】この構成においては、各ブロックは送受信するデータに基づいて最適な状態で（予め設定された基準の基づく）電源の供給や消費を行え、また、バスシステムを用いて統合的に電源供給・消費やデータの送受信を行える。したがって、種々のブロック間で電源供給・消費について融通し合うことができる。

【0011】また、バスシステムが、データバス管理のためにアドレス信号を周期的に発生し、そのアドレス信号に対応する情報は、バスに接続されるブロックが発信するようにしてもよい。

【0012】この場合、発生するアドレス信号が、システム上可能な最大数までとすることが好ましい。

【0013】また、発生するアドレス信号が、バス上にあるブロックの数+1としてもよい。

【0014】単一のデータバスラインを用いて、それに接続される複数のブロックのアドレスの設定および、データ通信の双方を行うようにすることが好ましい。

【0015】また、単一のデータバスラインを用いてそれに接続される複数のブロックのアドレスの設定およ

び、データ通信を二つの段階に分けて行うようにしてもよい。

【0016】単一のデータバスラインに複数のブロックが接続され、それら複数のブロックのアドレス付与の方式が以下の特徴を持つことが好ましい。アドレス設定開始タイミングがすべてのブロックに対して同時（半導体素子のばらつきや回路のディレイ等による差はここでは含まない）であり、各ブロックはランダムに発生したデータを同じタイミングでデータバスに出力し、従って、データバスライン上にはそれらのオア信号が発生し、データバスライン上の情報と各ブロックが発生した信号を比較する事で、他のブロックが発生したデータを知る事なく、一つのブロックが特定でき、特定されたブロックにユニークなアドレスを割り当て、この一連の動作を繰り返す事で最終的にはすべてのブロックにユニークなアドレスが割り当てられる。

【0017】また、アドレス付与後に、以下に示すユニークアドレス確認を行うことが好ましい。データバス管理手段が発生するアドレスに対し、そのアドレスの固有データをデータバス上に出力する事で、オア信号が発生し、各ブロックが自分の固有データとバス上のデータを照合する事で、自分以外に同じアドレスが存在する事を認識し、同じアドレスが存在するときには、再度、同位置のアドレスを有するブロックの範囲で、上述のアドレスバス付与を行ないアドレスを振り直す。

【0018】また、データバスライン上の特定アドレスに対するブロック属性が消滅した事でブロックの削除を認識し、逆にブロック属性が存在していなかった特定アドレスに対して、ブロック属性を新規検出する事で、ブロックの接続を認識するようにしてもよい。さらに、バスラインに接続された任意のブロックの間で、双方向のデータ通信が可能であり、かつ、この通信により任意のブロックが（自身を含む）任意のブロックの制御、情報読み出し、情報かき込みが可能なようにすることが好ましい。

【0019】

【発明の実施の形態】以下、本発明について詳細に説明する。

【0020】本発明は、ハードウェアとして、電源およびデータバスとそのバスコントローラ、それらバスに接続される電源構成ブロック、ソフトウェアとして電源構成ブロックのオブジェクトモデル化および、バスコントローラの動作メカニズム、電源構成ブロック間の通信フォーマットを包含する一つのアーキテクチャに関するものである。

【0021】ここでは内容が多岐にわたるため、次の3部構成で説明する。第1部：電源構成ブロックをどの様にオブジェクトモデル化するかについて説明。第2部：電源オブジェクトモデルを使用して実際の電源バスシステムをいかに構築するかについて説明する。第3部：構

10

20

30

40

50

築された電源バスシステム上で各オブジェクトモデルがどの様に動作するかについて説明する。

【0022】[第1部 電源に使用されるブロックのオブジェクトモデル化]

#### 1. 電源に使用されるブロックの整理

現在電源関係に使用されている、あるいは可能性のあるブロックは概略以下のものである。

##### (1) 電源供給源

: 一次電池 (乾電池等)

: ACアダプタ (商用ラインからDCを発生させ、機器の電源または二次電池の充電に用いる)

: DCアダプタ (カーバッテリー等のDC電源からターゲット機器に適するDCを発生させ、ACアダプタと同様の用途に用いる)

: 簡易発電機 (エンジン付の発電機から、手動発電機まで)

: 太陽電池

: 空間の電磁波からエネルギーを発生するもの (例えば、特開平10-146077号公報)

: その他各種燃料電池、生体からエネルギーを取り出すもの

##### (2) 電源消費者 (負荷)

: 各種機器 (一般に、この部分が実ユーザーに対して何らかの利便性を提供する)

##### (3) 電源供給、消費源

: 二次電池 (NiCd電池、Ni水素電池、リチウム電池、リチウムイオン電池、鉛電池等)

(電動自転車におけるモータの様に通常は電力を消費するものだが、制動時に電力発生の可能性をもつブロック (再生制動機能をもつもの) は、後述する理由により (3) ではなく (2) 負荷に分類するのが適当である。)

つまり、電源関係で使用されるブロックを、高いレベルで抽象化すると上記3つにわけられ、さらに (3) は

(1) と (2) の機能を共通にもつものと考えられる。またこれらを抽象化するには主としてコンピュータプログラミング分野で用いられている考えであるオブジェクトが有効である。従って、以下上記 (1)、(2)、

(3) をオブジェクトとして定義していくが、最初に簡単にオブジェクトとは何かについて説明する。

#### 【0023】2. オブジェクトの説明

オブジェクトとは、通常はコンピュータのプログラミング手法で用いられる用語であって、ある機能を実現する部分をブラックボックスとして扱い、ユーザ (プログラマ) にはその外部から見た仕様のみを与え、プログラミングの効率化を図る事を主目的とする。

【0024】オブジェクトは、その機能ブロックの仕様を示すデータ、そしてそれらのデータを変更したり、何らかの機能を実現する関数群から構成される。

#### 【0025】オブジェクト指向そのものは特定のプログ

ラム言語専有のものではないが、ここではもっとも一般的なC++言語により説明する。オブジェクトはC++においてはクラスとして定義され、例えば次の様なものである。(以下、山下 他著、C++プログラミングスタイル、p147、オーム社より引用)

【表1】

```
class Turtle{
public :
    int x;
    int y;
    int direction ;
    void forward(int);
    void turnleft();
    void turnright();
};

void Turtle::forward(int step)
{
    //前進の処理
}
以下略
(引用終了)
```

この例では、Turtle (亀) クラスを定義し、そのデータとして位置座標 x、y と方向 direction、クラスに対する動作を定義するものとして関数 forward(int), turnleft(), turnright() を定義している。つまり、Turtle というオブジェクトを、その現在位置、進行方向、左回り動作、右回り動作、前進動作というデータおよび機能で規定している。

【0026】3 電源に使用されるブロックのモデル化 (3-1) 電源システムに使用されるブロックの抽象化 図1は一般的な電源の供給、消費システムを示したもので、1、2、3はそれぞれ電源、GND、信号のためのバスラインである。3の信号バスについては、GND2を共通帰線とする単一の線の場合、電源ラインとは独立に複数の線により構成される場合、さらに電源バス

(1) に重畳され物理的には存在しない場合もある。4はバッテリー (ニッケル水素やリチウムイオンの様な二次電池、マンガンやアルカリ等の一次電池を問わない)、5は電気二重層コンデンサを貯蔵源とする様な一時電源、6はACアダプタ (チャージャと兼用の場合もある)、7は負荷となる一般電気機器である。これらバスに接続されるもの (以降でオブジェクトとして定義する) の数は問わない。いずれのブロックもパワーバスラインに必ずしもすべてが同時に物理的に接続されるものではなく、各ブロックとパワーバスラインとの間にスイッチ (実用的には半導体スイッチ) が挿入されている。

【0027】電源システムを構成する要素としては上記の様に各種考えられるが、これらの要素を抽象化すると、電源供給源となるもの（一次、ACアダプタ、DCアダプタ等）および、これらから電源を供給される消費者（負荷）に大別できる。オブジェクト指向の特徴の一つは、共通機能を持つ新しいオブジェクトをその元となるオブジェクトから派生させることが可能な点にある。従って、ここでもこのフィロソフィーに従い、まず本電源バスシステムに接続されるオブジェクトを定義し、それより電源供給源オブジェクトと負荷としてのオブジェクトを派生する。ACアダプタや一次電池などは電源供給源オブジェクトから派生させる。電気機器そのものは、本システムからみれば単に負荷オブジェクトとして抽象化される。また1で述べた（3）項に相当するものは電源供給源、および負荷の機能を継承して生成される。実用機器で主要な電源となる二次電池はこの（3）項に属すると考えるのが適当である。

【0028】また、二次電池に対する充電方式はそのバッテリーの種類（NiCdカリチウムイオンか等）により異なるし、電気二重層コンデンサを用いた一時電源の充電の仕方はバッテリーとも異なる。しかし、これらの相違点もオブジェクト指向の大きなフィロソフィーであるポリモーフィズムにより統一的に扱え、単にユーザ（この場合は、本アーキテクチャに基づいて電源関係機器を設計する設計者）からは同一のインターフェースで扱える。

【0029】（3-2）電源オブジェクト、負荷オブジェクトの投入、切断

図1に示すシステムにおいて、電源供給オブジェクト（図2）、負荷オブジェクト（図4）とも電源バスラインに対してスイッチを有している。電源供給オブジェクト、負荷オブジェクトとも1個の時は、それぞれに電源に対するスイッチを有するのは冗長であるが、複数の電源オブジェクトや負荷オブジェクト、さらには、電源または負荷となるオブジェクトを考慮するとこういう構造

にするのが適当である。

【0030】このような構造で、いったい誰がどのオブジェクトの（電源バスに対する）スイッチを投入するかについて以下のように規定する。電源供給オブジェクトのスイッチは、複数の電源供給オブジェクトが同じバスライン上に存在するときに、それらのうち一つのみを有効にするために用いる。複数の電源供給オブジェクトから一つを選択する際には、次のプライオリティ付けに従う。

【0031】

- 1 ACアダプタ
- 2 DCアダプタ
- 3 二次電池
- 4 一次電池
- 5 一時エネルギーストレージ

即ち、最も長時間エネルギーが供給できそうなものの順とする。また、同じレベルの電源供給オブジェクトが存在する場合は、例えばその容量順とかアドレス順とする（これはシステムの実装上の問題である）。負荷オブジェクトのスイッチ投入は、負荷の先に存在するセットの要求により制御されるのが原則であるので、負荷自身が制御する。この時、制御命令を自分自身に対して発信するようにする。こうすれば、バスライン上のほかのオブジェクトも情報共有ができる。

【0032】4 電源供給源ブロックのオブジェクトモデル化

（4-1）最上位オブジェクトモデル

以下、本実施例に示す電源システムに接続されるブロックのオブジェクトモデル化について説明する。

【0033】最初に、本電源システムに接続されるすべてのもの（バス管理手段を除く）を含むクラスを定義し、すべてのオブジェクトはこれから派生ないし継承で生成する。従って、このモデルがすべてのクラスの原型となる。

【表2】

```

typedef enum { POWER_SOURCE, LOAD, SOURCE_OR_LOAD } BlockType;
//電源ブロックを、供給源、負荷、供給源または負荷 の3種類に定義する。
typedef enum { POWER_SOURCE, LOAD } PowerMode ;
//電源のモード定義。 供給源 または 負荷。
typedef enum { FALSE, TRUE } boolean ;
//boolean の定義。
class Load;
class LoadList{           // 電源供給されている負荷リスト
public:
    Load* load ;
    LoadList* next ;
}

class PowerBlock {
private :
    BlockType block_type ;    // 電源供給ブロック、負荷、電源供給または負荷 を示すデータ
    PowerMode power_mode ;    // 電源供給モードか負荷モードかを示すデータ
    int address ;             // ブロックのユニークなアドレス
    boolean switch ;          // オブジェクトの電源バススイッチ状態、F=オフ、T=オン
public :
    BlockType BlockTypeInq(){ return block_type ;} // BlockType 問い合わせ関数
    PowerMode ModeInq(){ return power_mode ;}      // PowerMode 問い合わせ関数
    boolean SwitchCont( boolean cont){ return switch ;} // スイッチ制御関数
    boolean SwitchInq(){ return switch ;}          // スイッチ状態問い合わせ関数
};

```

20

上記クラスの定義の中で、データとしてアドレスは存在するが、関数としてアドレスを読み出すものはない。これは、本システムの構造上、オブジェクトに対してアドレスを問い合わせる事はできない（つまり、データ問い合わせのベースとなるのがアドレスであるので）からである。

【0034】（4-2）電源供給源のモデル化  
バッテリーやACアダプタ等の電源供給源の基本モデルとなるものである。このモデルは電源供給源として必要な情報、およびそれに関連する機能をまとめたものである。

【0035】図2において、10はDC電源であり、こ

れがバッテリーであるかACを整流した直流電源であるかは問わない。11は電流検出のための抵抗、12は電力バス（16）との接続を行うためのスイッチ、19はこの物理装置を論理的なオブジェクトとするためのマイクロプロセッサ（即ち、情報バスライン17から見込むとこの物理装置は論理的オブジェクトに見える）であり、具体的にはDC電源の電圧や出力電流の測定、スイッチ12のコントロール、自分自身のバス上のアドレス設定や他のオブジェクトとデータ通信を行う。

【0036】この様な一般的な電源供給源のオブジェクトモデル例を以下に示す。

【表3】

```

typedef int PowerType ;
class PowerSource : public PowerBlock {
private:
    // 仕様
    PowerType power_type ;    // 電源の種類 (一次電池、二次電池、AC アダプタ、DC アダプタ等)
    int catalog_voltage ;    // 公称出力電圧
    int max_current ;        // 公称最大出力電流
    int max_va ;             // 公称容量
    int alarm_voltage ;      // 電源最低供給可能電圧
    // 現在の状態
    int output_voltage ;     // 現在の出力電圧
    int output_current ;     // 現在の出力電流
    int current_imp ;        // 現在の内部インピーダンス
    int service_time ;       // 負荷の要求に対応できる予測時間
    int min_service_time ;   // 負荷の要求に対応する最低時間
    // 内部で使用するデータ
    LoadList *load_list ;    // 負荷リスト
public:
    // 内部データを読み出すための関数
    PowerType PowerType(){ return power_type ; }
    int CatalogVoltage(){ return catalog_voltage ; }
    int MaxVA(){ return max_va ; }
    int AlarmVoltage(){ return alarm_voltage ; }
    int OutputVoltage(){ return output_voltage ; }
    int OutputCurrent(){ return output_current ; }
    int CurrentImp(){ return current_imp ; }
    int ServiceTime(){ return service_time ; }
public :
    // オブジェクト制御のための関数
    boolean IsUsable( Load *l)    // 電源供給源が負荷オブジェクトにサービス可能か
    {
        引数は Load クラスへのポインターで、実質的に、負荷オブジェクトの、最低電圧、許容最高電圧、
        最小電流、最大電流等である。
        そして、関数としては、現在の電源供給源が
            負荷オブジェクトの許容最高電圧 > 予想出力電圧 > 負荷オブジェクトの要求最低電圧
            予想最大出力電流 > 負荷オブジェクトの要求電流
            最低供給電圧 (二次電池ならバッテリーダウン電圧) に達するまでの時間が、ある設定値以上か
        を評価し、False または True を返す。
    }
    boolean SetMinServTime( int t) // 最低サービスタイム設定
    {
        引数として設定時間を分単位でとり、正常設定なら True、エラーなら False を返す。電源供給オブ
        ジェクトに対して、最低限の動作時間を設定。これは主として一時ストレージ電源用コマンドであ
        る。
    }
    int RemainVA()                // 電源の残容量問い合わせ
    {
        電源の残容量でバッテリーな場合は電圧と電流より残容量を計算し、分単位で、AC アダプターや DC
        アダプターの場合ある決められた値 (例えば 0 x f f) を返す。
    }
};

```

また、以下に内部データに対するコードの割り当て例を示す。上記モデルだけでは二次電池は対象外であるが、二次電池は上記オブジェクトモデルと負荷オブジェクト 40 モデルを継承して生成されるので、PowerType のコード割り当てに二次電池や一時電源もいれておく。0 x は 16 進数を表す。

【表 4】

PowerType	code
二次電池、NiCd	0x00
NiH	0x01
LiIon	0x02
Lead	0x03
一次電池 Mn	0x04
Alkaline	0x05
AC アダプタ	0x06
DC アダプタ	0x07
発電機	0x08
一時電源	0x09

### 【0037】 (4-3) エネルギーチャージオブジェクト

これは、エネルギーストレージオブジェクトの充電、および電源供給を担当するものである。このオブジェクトはACアダプターであってもいいし、カーバッテリーから充電する様なDC入力のもの、あるいは、環境に存在する電磁波を電圧に変換する様な他の手段によるものでもいい。チャージャはそのターゲットとするものによって充電の方式までの異なるので、それらを一まとめにした一般的なモデル例を以下に示す。実際のオブジェクトはこれからさらに派生させて詳細データを定義していく。またそれら個々のチャージャはその実現方式に

```
class Charger :public PowerSource{
private :
    int charger_type ;           // チャージャの型 (Li イオン用、NiCd 用等の区別)
    int max_output_voltage ;     // 充電最大出力電圧
    int max_output_curr ;       // 最大充電電流
    int voltlim ;               // 出力電圧リミット値
    int currlim ;               // 出力電流リミット値
    boolean charge ;           // 充電中、非充電中
public :
    int SetVoltLim( int v ){ return voltlim ; } // 出力電圧リミット設定
    int SetCurrLim( int c ){ return currlim ; } // 出力電流リミット設定
    boolean ChargeCont( boolean cont ){ return charge ; } // 充電開始、終了
};
```

チャージャの型のコード割付例を示す。(Power Type の割付例参照)

【表 6】

charger_type	code
NiCd	0x00
NiH	0x01
LiIon	0x02
Lead	0x03
一時電源	0x09

### 【0038】 5. 負荷オブジェクトのモデル化

#### (5-1) 基本負荷オブジェクトのモデル化

基本的な負荷モデルに必要な情報および機能をまとめてモデル化する。実際のユーザにサービスを提供する製品は、本オブジェクト指向電源アーキテクチャにとっては負荷オブジェクトモデルの先に存在するもので、負荷オブジェクトの中に埋没し、それがどの様なものかは関与

よって、使用するコマンドやパラメータが異なる(例えばNiCd電池の充電においても、充電電圧の変化を検出する方式や、電池の温度変化を検出する方式、電池の状態を見ずに微少な電流で充電するいわゆるトリクル充電など種々の方式がある)ので、一般的にコマンド体系を定義するのは困難であるし、また適当ではない。そこで、この様な装置に固有なコマンドに対応するために、本発明報告のコマンド体系には、機器固有コマンドを送受するメカニズムが用意されている(第3部 7-3参照)。

【表 5】

しない。

【0039】図4において16、17、18はそれぞれ電源バス、データバスおよびGNDである。50は本来の負荷57(これが実際はユーザに利便性を提供する部分である)を入り切りするためのスイッチ、55は負荷オブジェクトを生成するマイクロプロセッサである。消費電流は抵抗51により検出する事を基本とするが、57の電流消費状況は既知であるので、その57を制御するマイクロプロセッサ58より通信路56を介してデータを入手してもいい。また、55と58が同一のプロセッサの場合も当然存在する。59は電源バス16の瞬断や、電源供給オブジェクトの交換時の急激な電圧変化を緩和するためのコンデンサで、電気二重層コンデンサや補助的二次電池等が使える。また省略可能な場合もある。

【0040】以下にオブジェクト例を示す。

【表 7】



```

typedef int LoadType ;
typedef enum { LOAD_OFF, LOAD_SUSPEND, LOAD_ON } LoadMode ;
Class Load : public PowerBlock {
private:
    LoadType load_type ;    // 負荷のタイプ、サスペンド可能等
    int max_voltage ;       // 最大動作可能電圧
    int min_voltage ;       // 最小動作可能電圧
    int max_curent ;        // 負荷 ON 時最大消費電流
    int min_current ;       // 負荷 ON 時最小消費電流
    int suspend_curr ;      // サスペンド時消費電流
    LoadMode load_mode ;    // 現在、負荷として接続されているか、否か。 サスペンドか
    int input_voltage ;     // 現在の入力電圧
    int input_current ;     // 現在の消費電流
public:
    LoadType LoadTypeInq(){ return load_type ;} // 負荷タイプ問い合わせ
    int MaxVoltage(){ return max_voltage ;}     // 最大負荷電圧要求
    int MinVoltage(){ return min_vltage ;}       // 最小負荷電圧要求
    int MaxCurrent(){ return maz_current ;}      // 最大負荷電流要求
    int MinCurrent(){ return min_current ;}      // 最小負荷電流要求
    int SuspendCurrent(){ return suspend_current ;} // サスペンド電流要求
    LoadMode LoadModeInq(){ return load_mode ;} // 負荷モード要求
    int InputVoltage(){ return input_voltage ;}  // 現在の入力電圧問い合わせ
    int InputCurrent(){ return input_current ;}  // 現在の入力電流問い合わせ
    int GetVA() {                               // 負荷容量問い合わせ
        引数 ; なし
        負荷の容量（電圧、電流）問い合わせ
        戻り値 ; 電圧 単位 V （上位4ビット）
                  電流 単位 100mA （下位4ビット）
    }
    int LoadCont( int cont){ // 負荷接続、切り離し
        引数 ; 0x00 負荷接続
              0x01 負荷切断
              0x02 負荷サスペンド
        戻り値 ; 0x00 正常動作
                 0x01 エラー
    }
    int GetCurrPowerConsmpt(){ //現在の消費電力計算
        引数 ; なし
        戻り値 ; 電流 X 電圧 、単位 VA
    }
};

```

#### 【0041】（5-2）発電負荷オブジェクト

例えば、最終負荷としてモータを考える。モータがトルクを発生している時は、負荷オブジェクトは純粋に負荷であって、そこから電力が生まれてくる事はない。しかしながら、モータを外部より強制的に回転させる力が働くとき（例えば、電動自転車で、モータには電力供給をせずに、慣性力により逆にモータを駆動するとき）には、負荷オブジェクトが回生制動をサポートしているならば、このオブジェクトは電源供給オブジェクトとも成り得る。しかしながら、この様なオブジェクトの実用化には次の問題がある。

【0042】即ち、電源バスシステムは、システムのアドレスポーリングタイミングに同期してのみ、電源供給オブジェクトの切り替えを行う事が可能で、システムに非同期に電力発生モードになる訳にはいかない。従って、該当オブジェクト内にこれを解決する手段が要るが、これは具体的には補助充電電池や電気二重層コンデン

サの様な蓄電手段を設けることになる。しかもこの補助電源による電力はこのオブジェクト内で使用する様な構築とするのが実用的であるので、単に負荷オブジェクトとして定義するのが最適である。

【0043】ただし、電源供給オブジェクト、負荷オブジェクトからの継承で生成できないと言う訳ではない。

#### 【0044】6. オブジェクトの継承

##### （6-1）二次電池のモデル化

二次電池や一時ストレージ等、ある時は電源供給源となるが、ある時は電力消費者（負荷）となるものは電源供給オブジェクトと負荷オブジェクトよりそれぞれの性質を継承して生成するのが適当である。

【0045】図2において、DC電源10をバッテリーと考えると、この図はそのままバッテリーオブジェクトの内部構成となる。この他にバッテリーの保護回路としてのヒューズや温度検出素子等、すでに実用化されている各種構成要素が含まれるが、本発明の趣旨に無関係

なので省略する。

【0046】以下にオブジェクトモデル例を示すが、これ以外の定義の仕方も存在するし、また新データや新機

能をつけ加える事もある。

【表 8】

```
class RechargeBatt : public PowerSource, public Load {
private:
    int ModelName ;           // バッテリーの型名
    int TotalLife ;           // バッテリーの寿命。 充放電回数。
    int TotalUsedTime ;       // 現在までの充電回数

public :
    int GetLifeTime(){        // 寿命計算
        引数 ; なし
        バッテリーの寿命を、残りの充放電回数として計算し、TotalUsedTime として保存
        戻り値 ; (TotalLife-TotalUsedTime)残り充電回数、 単位 × 10 回
    }
};
```

ここで、関数の中身（実装方式）は設計項目でありその具現化方式そのものは本発明とは直接の関係はなく、例えばバッテリー残量計算や表示アルゴリズムはすでにカムコードなどで実現されている。

【0047】バッテリーをこの様なクラスとして定義すると、負荷オブジェクトは以下の情報を知る事ができ

- {1} 現負荷に対してバッテリーが使用可能か
- {2} 現在のバッテリーの残容量
- {3} 現在使用しているバッテリーの（期待）寿命。

【0048】また、バッテリーの型名等は負荷オブジェクトには興味ない事がらであるが、バスラインに電源ブロックエミュレータを接続すれば、各ブロックの情報を読み出せ、サービス等に役立てる事も可能となる。従って、これらの補助情報をオブジェクトに持たせる場合もある。

【0049】（6-2）一時エネルギーストレージオブジェクト

これは、電気二重層コンデンサ等の大容量コンデンサを使用した、一時的に電圧を貯える事を主機能とするオブ

```
class TempStorage : public PowerSource, public Load {
private :
    int capacitance ;         // 電気二重層コンデンサの容量、単位μF
    int multiplier ;          // 指数
public :
    int Capacitance(){ return capacitance ;}
    int Multiplier(){ return multiplier ;}
};
```

【0052】[第2部 電源オブジェクトモデルによる電源バスシステムの構築]

#### 1. 基本システムの構築

前項までで、電源の各ブロックのオブジェクトモデル化は完了したが、これを実際のバスライン上にインプリメントするためには、さらに次の様な機能を定義する必要がある。

ジェクトである。図3にブロック図を示すが、DC電源の部分が電気二重層コンデンサ20に置き換わっただけで、他は図2と変わらない。マイクロプロセッサ19は14より電源が常に供給されるが、もちろん20のに貯えられている電力が放電すれば動作を停止する。そのため21の様に電力ラインから電力供給を受けるルートを用意しておく。また、電気二重層コンデンサ20は16を経由して充電されることも可能であるが、16を経由せずに、全く別の経路で外部より充電される場合（例えば、電磁誘導により電力バス16を介さずに）もある。

【0050】電気二重層コンデンサの容量は現在、0.1F程度から100F程度が容易に入手できるが、この部分は用途によっては通常の電解コンデンサも使用可能であり、この時は容量は10μF程度から、数千μFであろう。従って、容量範囲が広いのでμFを単位として、その係数を別途定義しておく。

【0051】これをオブジェクトとして抽象化したものを次の例に示す。

【表 9】

【0053】まず始めに、今回のバスシステム（電源、GND、データの3線式を仮定する）に電源の各ブロックを任意のタイミングで接続、切断を行う方式を説明する。

【0054】図5において、63はバスライン上に用意されたバス管理手段である。このバス管理手段は本報告でいうオブジェクトとは別のものである。この部分は実

際にはマイクロプロセッサで構成され、電源は電源バスから供給される。すなわち、バスライン上に電源供給オブジェクトが接続されない限り動作は開始しない。61はデータラインのプルアップ抵抗、62はマイクロプロセッサよりのアドレス管理データをデータバスに出力するためのもので、データラインは各電源オブジェクトとの間でワイアードオアとなっている。64は電源オブジェクトの一例である（ここでは、データラインがワイアードオアとなっているのを図示するのが目的なので詳細は省略する）。

【0055】バス管理手段63は、それに電源が供給され動作を開始すると図6に示す様に、ある決まった数N（Nの詳細については第3部、1、アドレスフォーマット参照）のアドレスパルスを、サイクリックに発生する。アドレスパルスと次のアドレスパルスの間にはそれらのアドレスに対応するデータをデータバス上に出力する時間tが割り当てられている。図7は実際のデータパルス波形を示す。図ではアドレスに4ビット分を割り当てているが、最初は常に0で、アドレスのスタートを表現するため、最大アドレス数は8である。なお、アドレス数の最大はもちろん8に制限されないし、またバス管理手段にはリチウム電池等のバッテリーをあらかじめ装着しておいてもいい。

【0056】以上の様な準備をしておいて、次に実際のアドレスの割り当ておよび通信の方式を示す。

#### 【0057】（1）アドレス設定モード

バスの状態には大きく分けて、各オブジェクトのアドレスを設定中の状態と、アドレスが確定してオブジェクト間でデータ通信を行っている状態がある。前者をアドレス設定モード、後者をデータ通信モードと定義する。アドレス設定モードは後述する様にアドレス設定プロセスとアドレス確認プロセスから構成される。

【0058】アドレス設定モードは負荷オブジェクトがすでに接続された電源バスシステムに電源供給源オブジェクトが始めて接続された場合や、オブジェクトからエラー信号が発生され、再度電源システムの再構築が必要な場合に発生する。バスシステムがデータ通信モード中に新規オブジェクトが追加あるいは取り外しをされた場合には、アドレス設定はデータ通信モードの中で可能なので、アドレス設定モードとならない。（ただし、アドレス設定モードにする様にインプリメントする事はもちろん可能である。）詳細は後述する。

#### 【0059】（1-1）アドレス設定プロセス

バスライン上に接続されたオブジェクトにアドレスが定義されていない状態で、電源が入りアクティブになったとき、最初に各オブジェクトにアドレスが定義される。これをアドレス設定プロセスと呼ぶ事にする。

【0060】ここで実施しようとしているのは、全く同じタイミングに全く同じアルゴリズムを持った各オブジェクトが、一本のデータバスラインを共有しながら、そ

れぞれを区別しようとするものである。このためには、共通のバスラインに何個のオブジェクトが接続されているかの判定と、それらの区別が必要である。これを例えばアナログ的に実施するならば、プルアップされたデータラインに各オブジェクトが同一のインピーダンスでプルダウンし、データラインのDCレベルを観測する方式が考えられる。（すなわち、オブジェクトの数に対応したDCレベルが観測される。）これを認識し、オブジェクトを一つずつ切り離していき、最後に残ったものに、まず一つのアドレスを与え、その既存アドレスを得たオブジェクト以外のオブジェクトに対して同じ事を繰り返す。この場合でも、同一アルゴリズムを持ったオブジェクトでは、どれを切り離すかの判定が（各オブジェクト自身）できない。そのため、各オブジェクトに乱数を発生させる機能を用意し、その値によってどのオブジェクトを切り離すか（あるいは残すか）をオブジェクト自身に判断させる。つまり、オブジェクト間で一種じゃんけんをして決める方式である。

【0061】このような考えを、完全にデジタル的に実施したものを以下に述べる。

【0062】ここでは、最大8個の電源オブジェクトのアドレス設定について説明するが、もちろんこの数は8に制限される訳ではない。図8に示すようにすべてのアドレスパルスのあとには固定長1の後b0からb7まで8ビット分のアドレス投票区間が用意される。バスラインに接続されているすべての電源オブジェクトは、8ビット長のうちただ1ビットのみが“1”であるパルス（即ち、データとしては1, 2, 4, 8, 16, 32, 64, 128のうちの一つ）をランダムに発生し、この区間で同時に投票（出力）する。

【0063】図9はその様な投票例であり、オブジェクト1から8までがそれぞれ、b6、b4、b5、b3、b2、b0、b0、b7に投票した場合である。従って、これらのオア出力は最下段に示す様に、11111101となる。さて、各オブジェクトはこのオア出力と自分が投票した出力をたよりにアドレスを設定していく。図9のオア出力をみれば、LSB側に一番近いビットで“1”になった場所（この場合はb0でLSBそのもの）がわかる。ここに投票したオブジェクトをアドレスmに設定するため、次のアドレスパルス発生タイミング（この時mをインクリメントしてもかまわないが、ここでは簡単のため最終的にアドレスが決定されるまでは、アドレスのインクリメントは行わないとする。つまりアドレスmが設定完了するまで、アドレスmがバス管理手段により出力される。）の投票権をこれらのオブジェクトだけに与える（投票権を自分が有しているかどうかは各オブジェクト自身を知る事ができ、しかも他のオブジェクトがどう投票したかについて一切知らなくていい点が重要である）。この場合オブジェクト6と7が投票権を得て、図10に示す様に、再度投票する。この時

は b 2、b 4 にそれぞれ投票し、オア出力は 0 0 0 1 0 1 0 0 が得られた。ここでオア出力にはまだ 2 ビット以上 “1” が存在するので、以前と同様に L S B 側に一番近いビットで “1” になった場所に対応するオブジェクトが次の投票権を得る。この投票の様子を示したのが図 1 1 で、ここではオブジェクト 6 のみが投票をし、オア出力とオブジェクトの投票ビット位置が一致する。この段階で暫定的にオブジェクト 6 にアドレス m を付与する。

【0064】ここで暫定的というのは、各オブジェクトが発生する投票場所は全く確率的なもので、たとえば図 9 でオブジェクト 6、7 が同じビットに投票すると、6、7 に同じアドレス m を付与してしまう。そこで、こ

バイト数	宛先アドレス
3	ブロードキャスト (0 x 7 f)

【0067】(1-2) アドレス確認プロセス  
オブジェクトにアドレス m が暫定割り当てされたならば、アドレス確認プロセスに入る。このプロセスでは、バス管理手段により発生させられたアドレス m に対して、それぞれ自分のデータを出力し、データバス上の情報 (オアデータ) と自分自身のデータを比較する。(図 1 2 参照)

バイト数	宛先アドレス	コマンド
3	ブロードキャスト (0 x 7 f)	0 x 0 1

各パケットは、この情報により現在のバスのモードを知る事ができ、暫定アドレスを取得したオブジェクトは、自分の暫定アドレスに続いて自分のデータを出力する。

バイト数	宛先アドレス
4	自分のアドレス

宛先アドレスは自分自身のアドレスであり (この様に決めておく)、コマンドはアドレス確認プロセスである事を示す値 (0 x 0 1) であるが、パラメータに関しては、自分に固有のものを出力する。例えば、パラメータとしてシリアル番号を出力すれば、二つ以上のオブジェクトの固有データのオアを見ることで、自分以外に同時に出力した別オブジェクトの存在が認識できる。

【0069】ここで、アドレス重複が検出されたならば、再度アドレス設定プロセスを通過させる。

【0070】上記 (1-1)、(1-2) のアドレス設定、確認プロセスのフローチャートを図 1 3 に示す。なお、図 1 3 の動作はその記述内容から容易に理解できるので詳細な説明は省略する。

【0071】(2) データ通信モード

上記 (1-1)、(1-2) 項のプロセスを経る事で、各電源オブジェクトのアドレスは一義的にきまる。この後バス管理手段プロセッサは、アドレスを定期的に発行し、それに対応するオブジェクトは自分の存在を示すため、データをバスライン上に出力する。これにより、ア

の問題をさけるためのアドレス確認プロセスを入れるが、これについては後述する。

【0065】さて、確認プロセスも通過して、上記の様に、オブジェクト 6 のアドレスが m に確定すると投票権が消滅し、オブジェクト 6 を除く 1 から 8 について、アドレス m+1、m+2 … において同様の操作を行う。これを繰り返す事で、最終的にすべてのオブジェクトのアドレスが確定する。

【0066】アドレス設定プロセスである事は、バス管理手段がアドレス 0 のタイミングで発行するデータ中の値で判別する (コマンド=0 x 0 0)。詳細は後述する。

【表 1 0】

コマンド
0 x 0 0

アドレス確認プロセスであることはバス管理手段のアドレス (0) の後にバス管理手段より発行される以下のデータにより判別できる (コマンド 0 x 0 1 がアドレス確認プロセスを示す)。(データのフォーマットについては第 3 部参照。)

【表 1 1】

コマンド
0 x 0 1

【0068】この時のフォーマットはつぎのとおりである。

【表 1 2】

コマンド	パラメータ
0 x 0 1	固有値

ドレスが一巡すると、バスライン上にあるすべてのオブジェクトは他のオブジェクトの属性とアドレスを共有する。また、このモードではオブジェクト間同士の通信が行われる。この状態をデータ通信モードと定義する。データのフォーマット等詳細については 3 部で詳述する。

2 オブジェクトの追加、削除 (システムの動的再構築)

オブジェクトが電源バスに追加されたときに、そのデータラインを見る事によって、現在他のアドレスがすべて確定しているか、あるいは確定中かがわかる。すべて確定しているならば、空いているアドレス (つまり、アドレスのあとのデータ部分が空) に続けて、自分のデータを出し始めるだけでいい。バス管理手段プロセッサや、他のオブジェクトは、アドレスのあとにデータが入ってきたのを見て、オブジェクトの追加を認識する。確定中ならば、確定するまで待つて、その後同様の動作を開始する。

【0072】これ以降は各オブジェクトが必要に応じて、情報の要求、自発的送信を行う。

【0073】オブジェクトの削除により、そのアドレスに対応するデータが無くなる（ただし、唯一の電源供給オブジェクトが削除されたときは、システムはそのまま停止する）と、各オブジェクトはこれにより、そのアドレスに付随するオブジェクトの消滅を検出する。この時、管理機能プロセッサはアドレス詰めを行ってもいいし、とりあえずそのアドレスを空きにしておいてもいい（アドレスを現在のオブジェクト数+1だけポーリングする実装では、アドレス詰めが必須。しかしすべてのアドレスをポーリングする実装ではどちらでもいい）。い

【0074】〔第3部 電源バスシステムの制御体系〕  
これまでに、第1部で電源を構成する主要ブロックのオブジェクト化について述べ、第2部で実際の電源システムが如何に構築、維持され、また動的に再構築されるかの概要を説明した。第3部では、これらの構築、維持に使用されるコマンド体系（API群）およびそのフォーマットについて説明する。

【0075】なお、以下に示すコード類で総バイト数や宛先アドレスについて、チェックサムを別にしているのに注意されたい。

【0076】1 アドレスフォーマット

各オブジェクトのアドレスはバス管理機能プロセッサよりサイクリックに発生され、その長さは7ビット（従って、最大アドレス数は128であるが、バス管理プロセ

宛先アドレス（1バイト） コマンド（1バイト） パラメータ（1バイト）  
このパケットが一つのアドレスのデータ部分に、最小1個（3バイト）、最大127個（381バイト）存在する。この3バイトを一単位とする総パケット数を各オブジェクトおよびデータバス管理手段プロセッサが発信するデータの先頭に置く。（図14の101）この値を見ることで、何個のコマンドパケットが存在するか分かつとともに、この数でパケットの同期を取る。

【0082】3 アドレス発生周期の最大値

図15において101以降が、各オブジェクトが発信するデータである。

【0083】その基本単位は最小3バイトによるパケットであり、パケット数、通信先のアドレス、通信先に対するコマンド、そしてコマンドの引数（パラメータ）より構成される。

【0084】アドレスで指定されたオブジェクトがいくつかのパケット数を送信するかを示すのが、101のパケット数表示バイトであり、LSBをチェックサムとしているため、最小1、最大127である。

【0085】図16に一サイクルの通信形態と、100kbpsの速度で通信した場合の概略時間を示す。情報パケットの最小は3バイト（パケット数、宛先アドレス、コマンド）であるから、同期のための空白ビット等

ッサのアドレスを0、ブロードキャストアドレスを127と定義する）。このアドレスの間に各オブジェクトが自分のデータをバスラインに乗せる事ができる。この部分のフォーマットを図14に示す。

【0077】図において100はバス管理機能プロセッサが発生するアドレスであり、先頭（MSB）は常に0、そのあと7ビットのアドレスが続く。なお、このアドレス部はアドレス設定モード以外は、単純にインクリメントされていくのでチェックサムは用意していない。

【0078】このアドレスの値については次の規定を設ける。

アドレス0（0x00）；これはバス管理機能プロセッサのアドレスと定義する。

アドレス127（0x7F）；これはブロードキャストアドレス。すべてのオブジェクトはこれに続くデータに反応するようになっている。

【0079】従って、実際にオブジェクトに使用可能なアドレスは1から126までの126個である。

【0080】また、バス管理手段が発生するアドレスは0から126まですべてサイクリックに発生する方式と、アドレス設定モード内のみ、すべてのアドレスを発生し、データ通信モードでは実際に接続されているオブジェクトの数+1（つまり、新規追加オブジェクトの追加空きスペース）として周期を短くする方式がある。

【0081】2 パケットの最大値

1パケットを次の様に定義する。

【表13】

のオーバーヘッドを無視すると

最小 4x127（最大アドレス数）=509バイト

最大 (1+3x127)x127=48.5Kバイト  
となり、100Kbpsの通信速度なら、一サイクルは  
最小 41mS

最大 約3.8秒

となる。最大値は、すべてのアドレスをポーリングし、すべてのアドレスがおのおの127バイトのデータを有するときの値であるので、実用的にはこれより十分短い。

【0086】4 コマンド発生形態

図17に各オブジェクトからのコマンドの発信状況および、コマンドに対する回答状況を示す。すべてのコマンドの発信とその回答は1対1になる様に設計され、必ずハンドシェークとなる。アドレスの発生周期1、2、3はこの順に連続している。例えばオブジェクト1（アドレス1をアサインされたオブジェクト）がnに対してコマンドを発行（204）すると、オブジェクトnはこの内容を発行直後に（つまり、例えば206のタイミングまで待たずに）認識できる。そこでこれに対する回答を用意しておき、自分のデータ発信タイミング（これが206）で実際に情報発信を行う。また、オブジェクトn

は複数のアドレスにデータを送信できるので、例えばオブジェクト 2 に対する回答 (207) やオブジェクト 3 に対するコマンド発信 (208) が、自分の割り当てタイミング (つまりアドレス n を受け取ったあと) の中で可能である。

【0087】命令や情報要求 (これらはすべてコマンドと総称する) に対する、動作完了や回答時に使用するコマンドは、返答コマンドとして定義した。(コード 0x26)

また、同一宛先に対して複数の命令を送る事も可能で、この時は宛先アドレス-コマンド-パラメータのシーケンスを繰り返し、また回答はコマンド順に発生すると定義する。(212、213)

コマンドに対する回答のペアはネストを禁止してあり、コマンド-回答の間でエラーが発生した場合には、再度コマンドを発行する等で対処する。また、コマンドを自

アドレス設定モード	バイト数	宛先アドレス	コマンド
アドレス設定プロセス	3 ( 0x03	ブロードキャスト 0x7f	0x00 0x00)
アドレス確認プロセス	3 ( 0x03	ブロードキャスト 0x7f	0x01 0x01)
データ通信モード	3 ( 0x03	ブロードキャスト 0x7f	0x7f 0x7f)

これらのモードの遷移は一サイクル (すべてのアドレスポーリング完了まで) 単位となる。

【0091】6 オブジェクト属性発信体系

電源バスがデータ通信モードになると、各オブジェクト

宛先アドレス	コマンド	パラメータ	
0x7f	0x20	0x01	Generic Power Source
		0x02	Generic Charger
		0x03	AC adaptor
		0x04	DC adaptor
		0x05	NiCd battery
		0x06	NiH battery
		0x07	Li ion battery
		0x08	Lead battery
		0x09	Other battery
		0x10	Generic Temp Power Storage
		0x20	Generic Generator
		0x80	Generic Load

各オブジェクトはアドレスとそれに付随するコマンド、パラメータを読む事により、現在の電源バスライン上にどのような属性のものが接続されているかが知れる。さらに詳しい情報は必要なオブジェクトが相手先に問い合わせ、認識できる。

【0092】7 コマンド体系および使用例

本明細書はコマンド体系全体についての仕様書ではない

return value	API	Code	Parameter	Description
void	AdSetProces	0x00	padding	Address is under negotiation
void	AdConfirmProces	0x01	padding	Address is under confirmation
void	DataComMode	0x02	padding	Data communication mode

分自身に対して発行する事も可能であり、例えば負荷オブジェクトをオンするコマンドを負荷オブジェクト自身が発行すれば、バスラインに接続されている他のオブジェクトもこの情報が共有できる。これにより電源供給源オブジェクトは、電流が増えた事を検出せずに、負荷が接続されて事が知れる。

【0088】5 バスのモード電源バスには上述した様に、アドレス設定モードとデータ通信モードがあるが、この違いはバス上で以下の様に表明される。

【0089】アドレス 0 はバス管理機能プロセッサのアドレスであり、その後に最小 3 バイトのデータパケットが存在する。(この形式は他のオブジェクトと同じ)。

【0090】このデータパケットを用いて、上記モードの表明を行う。

【表 14】

は自身の属性をブロードキャストする。この時のデータの形式は以下の様になる。

【表 15】

ので、いくつかの使用例におけるコマンド (API) 群について記述する。なお、ここでいうコマンドは広い意味であり、内容的には情報要求命令、要求情報回答、動作命令、モード表明等を含んでいる。

【0093】(7-1) データバス管理コマンド  
データバス管理コマンドとしてはつぎのものがある。

【表 16】

【0094】(7-2) データ通信モード中に使用されるコマンド

データ通信モード中に使用されるコマンドとしてはつぎ

return value	API	Code	Parameter	Description
BlockType	BlockTypeInq	0x10	padding	Inquiry for power block type
PowerMode	ModeInq	0x11	padding	Inquiry of power source or load
boolean	SwitchCont	0x13	boolean	Switch on/off
boolean	SwitchInq	0x14	padding	Inquiry of switch on/off
PowerType	PowerType	0x15	padding	Inquiry of power source type
int	CatalogVoltage	0x16	padding	Inquiry of nominal voltage
int	MaxVA	0x17	padding	Inquiry of maximum VA
int	AlarmVoltage	0x18	padding	Inquiry of alarm voltage
int	OutputVoltage	0x19	padding	Inquiry of output voltage
int	OutputCurrent	0x1a	padding	Inquiry of output current
int	CurrentImp	0x1b	padding	Inquirey of output impedance
int	ServiceTime	0x1c	padding	Inquiry of expected service time
boolean	IsUsable	0x1d	pointer to Load	Inquiry of power source usable
boolean	SetMinServTime	0x1e	time	Set minimum service time
int	RemainVA	0x1f	padding	Inquiry of remained power
int	GetLifeTime	0x20	padding	Inquiry of remained lifetime
int	Capacitance	0x21	padding	Inquiry of capacitance
int	Multiplier	0x22	padding	Inquiry of unit multiplier
int	SetVoltLim	0x23	voltage	Set limit voltage
int	SetCurrLim	0x24	current	Set limit current
boolean	ChargeCont	0x25	boolean	Charge on/off control
int	DataReply	0x26	reply data	Reply data

20

のようなものがある。

【表 17】

【0095】(7-3) 例外コマンド

本アーキテクチャのフォーマットに準拠しない、固有のデータやコマンド等を通信するためのラップとして次の

return value	API	Code	Parameter	Description
**	Wrapper	0x70	**	Command wrapper
**	ReplyWrapper	0x71	**	Reply wrapper

ものを定義する。

【表 18】

以下、実際の応用例である

バケット数	宛先アドレス
m	1 バイト
n	1 バイト

30

【表 19】

コマンド	パラメータ
0x70	固有情報
0x71	固有リブライ情報

パラメータ中に固有情報を挿入するが、その長さはmバケット(3mバイト)となるように調節し、最大127バケット(3x127バイト)である。

(第1部4-1参照) 負荷オブジェクト(アドレス8)がバスライン上の電源供給オブジェクト(アドレス2)に対し、その詳細情報要求例を示す。

【0096】(7-4) データ通信モード中での他のオブジェクトに対する情報要求例

【表 20】

address	bytesize	destination address	command	parameter
0x08	0x04	0x02	0x14	0x00(padding)
0x02	0x04	0x08	0x15	0x00

40

ここに、コマンド0x14はオブジェクトに対するスイッチデータ情報要求、パラメータ0x00は単にスペースを埋めるためのもの、電源供給オブジェクトが発行したコマンドは0x15でコマンドに対する返事を表現し、パラメータ0x00は、スイッチがオフである。

【0097】8 クラス一覧以下、クラスの一覧を示す。

【0098】(8-1) 基本クラス  
基本クラスはつぎのように記述される。

【表 21】

```

class PowerBlock {
private :
    BlockType block_type ;
    PowerMode power_mode ;
    int address ;
    boolean switch ;
public :
    BlockType BlockTypeInq(){ return block_type ; }
    PowerMode ModeInq(){ return power_mode ; }
    boolean SwitchCont( boolean cont){ return switch ; }
    boolean SwitchInq(){ return switch ; }
};

```

【0099】 (8-2) 電源供給オブジェクト

【表22】

電源供給オブジェクトはつぎのように記述される。

```

class PowerSource : public PowerBlock {
private:
    PowerType power_type ;
    int catalog_voltage ;
    int max_current ;
    int max_va ;
    int alarm_voltage ;
    int output_voltage ;
    int output_current ;
    int current_imp ;
    int service_time ;
    int min_service_time ;
    LoadList *load_list ;
public:
    PowerType PowerType(){ return power_type ; }
    int CatalogVoltage(){ return catalog_voltage ; }
    int MaxVA(){ return max_va ; }
    int AlarmVoltage(){ return alarm_voltage ; }
    int OutputVoltage(){ return output_voltage ; }
    int OutputCurrent(){ return output_current ; }
    int CurrentImp(){ return current_imp ; }
    int ServiceTime(){ return service_time ; }
public :
    boolean IsUsable( Load *l) { ; }
    boolean SetMinServTime( int t) { }
    int RemainVA() { }
};

```

【0100】 (8-3) チャージャ

【表23】

チャージャはつぎのように記述される。



```

class Charger :public PowerSource{
private :
    int charger_type ;
    int max_output_voltage ;
    int max_output_curr ;
    int voltlim ;
    int currlim ;
    boolean charge ;
public :
    int SetVoltLim( int v ){ return voltlim ; }
    int SetCurrLim( int c ){ return currlim ; }
    boolean ChargeCont( boolean cont ){ return charge ;}
};

```

【0101】(8-4) 負荷

【表24】

負荷はつぎのように記述される。

```

Class Load : public PowerBlock {
private:
    LoadType load_type ;
    int max_voltage ;
    int min_voltage ;
    int max_curent ;
    int min_current ;
    int suspend_curr ;
    LoadMode load_mode ;
    int input_voltage ;
    int input_current ;
public:
    LoadType LoadTypeInq(){ return load_type ;}
    int MaxVoltage(){ return max_voltage ;}
    int MinVoltage(){ return min_vltagae ;}
    int MaxCurrent(){ return maz_current ;}
    int MinCurrent(){ return min_current ;}
    int SuspendCurrent(){ return suspend_current ;}
    LoadMode LoadModeInq(){ return load_mode ;}
    int InputVoltage(){ return input_voltage ;}
    int InputCurrent(){ return input_current ;}
    int GetVA() {    ;}
    int LoadCont( int cont){    ;}
    int GetCurrPowerConsmpt(){    ;}
};

```

【0102】(8-5) 二次電池

【表25】

二次電池はつぎのように記述される。

33

34

```

class RechargeBatt : public PowerSource, public Load {
private:
    int ModelName ;
    int TotalLife ;
    int TotalUsedTime ;
public :
    int GetLifeTime() ;
};

```

10

【0103】(8-6) 一時電源

【表 26】

一時電源はつぎのように記述される。

```

class TempStorage : public PowerSource, public Load {
private :
    int capacitance ;
    int multiplier ;
public :
    int Capacitance(){ return capacitance ;}
    int Multiplier(){ return multiplier ;}
};

```

つぎに、電源および負荷が1つずつの最も基本的な例を用いて具体的に説明する。

【0104】図18において、バスコントローラ301に電源バスライン302、GNDバスライン303およびデータバスライン304が接続されている。これらバスラインにコネクタ307および308を介して電源305および負荷306がそれぞれ接続されている。

【0105】各部の動作状況を説明する。

【0106】(1) 電源の内容

PowerSourceクラスに属する電源の仕様は以下のものであるとする。

power\_\_type = 0x00 (二次電池でさらにNiCd電池であることを示す)

catalog\_\_voltage = 6 (公称出力電圧 = 6V)

max\_\_va = 6 (公称容量 = 6VA)

output\_\_voltage = 7 (現在の出力電圧 = 7V)

alarm\_\_voltage = 5.6 (最低供給可能電圧 = 5.5V)

上記データの中で、現在出力電圧が公称出力電圧より高いが充電直後はこのようになるのが普通である。また最低供給可能電圧5.5Vは、例えば、これより低い電圧になるまで電池が使用されると電池が痛むという値である。

【0107】(2) 負荷の内容

一方loadクラスに属する負荷のデータは以下のものと仮定する。

max\_\_voltage = 8V (入力許容最大電圧 = 8V)

min\_\_voltage = 4V (動作可能最小電圧 = 4V)

max\_\_current = 1A (入力最大電流 = 1A)

【0108】(3) 電源システムの動作

図18に示すようなブロックで、電源305がバスに接続された後、負荷306がバスに接続される場合を想定する。(負荷が先に接続されても、何も起こらない) 図2に示すように電源オブジェクトの中のマイクロプロセッサ19はDC電源10がある値以上であればつねに動作している。

【0109】従って電源305がバスに接続されると、バス管理手段(バスコントローラ301)が動作を開始し、まずアドレス設定モードでシステムが動作する。これにより図13に示すフローチャートに基づき、電源オブジェクト(電源305)のアドレスが決定する(アドレス1となる)。

【0110】アドレスが決定された時点で、このバスには他に電源が接続されていない事(実際には他には何も接続されていない事)がわかるので、バスシステムを使用可能とするため、電源オブジェクトのメインスイッチ(12)を投入する。

【0111】つぎに負荷オブジェクト(負荷306)が接続されると、再度アドレスの調停を行い、負荷オブジェクトのアドレスが決定(アドレス2)する。

【0112】以上のプロセスを経て、電源バスシステム

は動作準備が整う。

#### (4) 実際のオペレーション例

電源バスシステムがどの様に使われるかは本発明の趣旨であるアーキテクチャの上のアプリケーションプログラムで決定されるものであるが、以下に一例をしめす。

【0113】負荷として上記の様に（上記（2）項）仮定したが、実際の状況ではこの負荷の先にラジオであるとか、PDAであるとかの製品が存在する。ここでは一番簡単にするため、オンオフ機能のみ付いている製品で例えばフラッシュライトとする。

【0114】フラッシュライトの場合の負荷オブジェクトの構造を図19に示す。309がマイクロプロセッサ、310はメインスイッチで、例えばMOSFET、312はランプ、311がランプのコントロールスイッチ、313はアラーム表示LEDである。

【0115】コネクタ308がバスシステムに接続され、負荷オブジェクトのアドレス調停が完了した時点で、マイクロプロセッサは次の様な動作を始める。この動作を図20に示す。なお、図20の動作は基本的に図

【0116】電源電圧の問い合わせ；Catalog Voltage（）関数を使用して電源オブジェクトに対して公称出力電圧を問い合わせ、これと自分自身の最大許容電圧を比較し、電源電圧が適正かどうか判断する。公称電圧はもちろん実際電圧とは異なるが、これがわかればおよその判断は可能である。もし、非常に厳密な判断が必要なら、そのようなデータや問い合わせAPIを定義すればいい。

【0117】電源の種類問い合わせ；電源電圧は適正でも自分が必要な電流容量を判断するために、電源の種類から判断する。フラッシュライトで最大1Aならば、ほとんどの一次、二次電池は使用可能である。これも電流容量が正確に知りたいならばそのようなAPIを用意すればいい。

【0118】ここまでで、電源の電圧とおよその供給電流が知れたので、フラッシュライトのコントロールスイッチを検出する。ここでスイッチがオンされちならば、電球112に電力を供給する。この後はバッテリー残量を適宜問い合わせ（フローチャートでは、この辺の時間のウェイトをかける部分は省略）、バッテリー残量が少なくなったならば、電球を切り、アラームLEDを点灯する。

【0119】アラームLEDは、フラッシュライト使用中にバッテリーが少なくなった時のみでなく、バスシステムに接続されている電源との相性が悪いときにも点灯させて、ユーザに状況を表示するためのものである。

#### 【0120】

【発明の効果】以上説明したように、この発明によれば、以下のような効果を実現できる。1 モバイル機器、

ウェアラブル機器に対して、共通の電源管理アーキテクチャを提供し、バッテリーや電源の共通化がはかれる。

2 異なった種類の電源（一次電池、二次電池、ACアダプター、DCアダプター、発電機）や異なった機器

（負荷）の間で、お互いの状況を通信する共通の通信手段および言語体系を提供し、複数の機器間で共通に使用可能な電源関連機器設計ができる。

3 各種の電源供給源を共通のアーキテクチャの中で定義する事により、幅広いエネルギー供給源を機器の電源に使用可能となり、機器使用環境が飛躍的に拡大する。

4 複数の異なった種類の電源が共通のバスラインに論理的に並列接続が可能となり、バスラインに接続する電源の数を増やすだけで、電源容量を上昇できる。

5 本電源管理アーキテクチャはスケラブルであり、比較的小電力のものから、オブジェクト数が多く、かつ大電力を扱うものまで同一の考えで設計できる。

6 本アーキテクチャに基づいて設計する事により、電源関係ブロックの標準化が可能で、電源システム設計の大幅な省力化が可能である。

7 オブジェクトエミュレータにより、例えばPCなどにより電源の動作状態をモニターする事ができ、信頼性の高い電源システムを容易に構築できる。

#### 【図面の簡単な説明】

【図1】 この発明の実施例の構成を全体として示す図である。

【図2】 上述実施例の電源オブジェクトの構成例を説明する図である。

【図3】 上述実施例の一時エネルギーストレージオブジェクトの構成例を説明する図である。

【図4】 上述実施例の負荷オブジェクトの構成例を説明する図である。

【図5】 上述実施例のバス管理手段を説明する図である。

【図6】 上述実施例のアドレス付与を説明する図である。

【図7】 上述実施例のアドレス付与を説明する図である。

【図8】 上述実施例のアドレス付与を説明する図である。

【図9】 上述実施例のアドレス付与を説明する図である。

【図10】 上述実施例のアドレス付与を説明する図である。

【図11】 上述実施例のアドレス付与を説明する図である。

【図12】 上述実施例のアドレス付与を説明する図である。

【図13】 上述実施例のアドレス付与を説明する図である。

【図14】 上述実施例のデータ通信フォーマットを説

明する図である。

【図 15】 上述実施例のデータ通信フォーマットを説明する図である。

【図 16】 上述実施例のデータ通信フォーマットを説明する図である。

【図 17】 上述実施例のコマンドの発信、回答を説明する図である。

【図 18】 上述実施例を具体例をあげて説明する図である。

【図 19】 図 18 の負荷の構成を説明する図である。

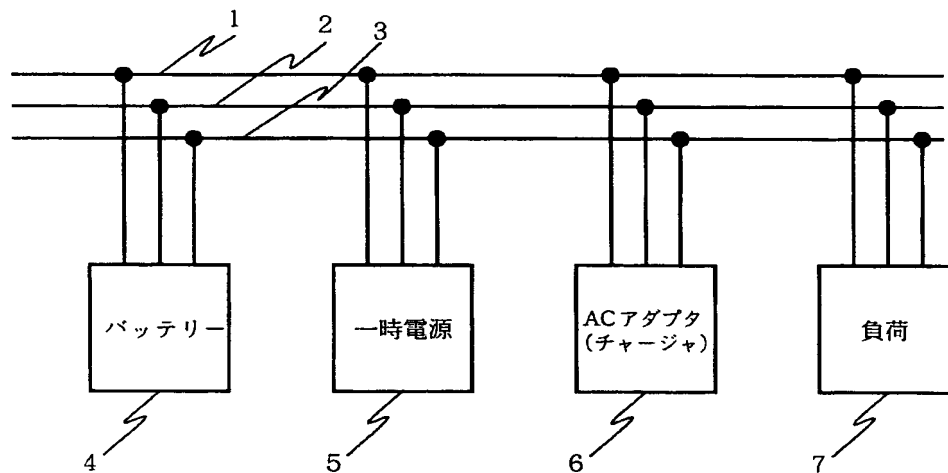
【図 20】 図 18 および図 19 の動作を説明する図である。

【符号の説明】

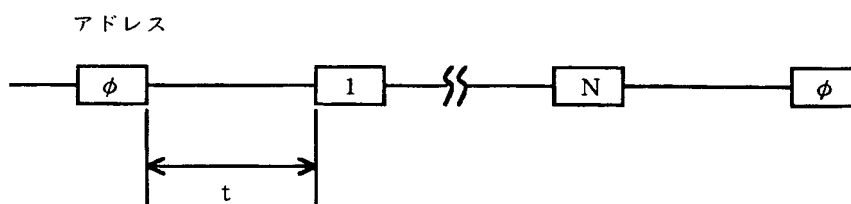
- 1 電源バスライン
- 2 GND (接地) バスライン
- 3 データ (信号) バスライン
- 4 バッテリー
- 5 一時電源
- 6 ACアダプタ
- 7 負荷 (一般電気機器)
- 10 DC電源

- 11 電流検出用抵抗
- 12 スイッチ
- 13、14、15、21 マイクロプロセッサの入出力
- 16 電力バスライン
- 17 情報バスライン
- 18 接地バスライン
- 19 マイクロプロセッサ
- 20 電気二十層コンデンサ
- 50 スイッチ
- 51 消費電流検出用抵抗
- 52、53、54 マイクロプロセッサ 55 の入出力
- 55 マイクロプロセッサ
- 56 通信路
- 57 本来の負荷
- 58 負荷 57 のマイクロプロセッサ
- 59 コンデンサ
- 60 マイクロプロセッサ
- 61 プルアップ抵抗
- 62 データ出力用トランジスタ
- 20 63 バス管理手段 (バスコントローラ)
- 64 電源オブジェクト

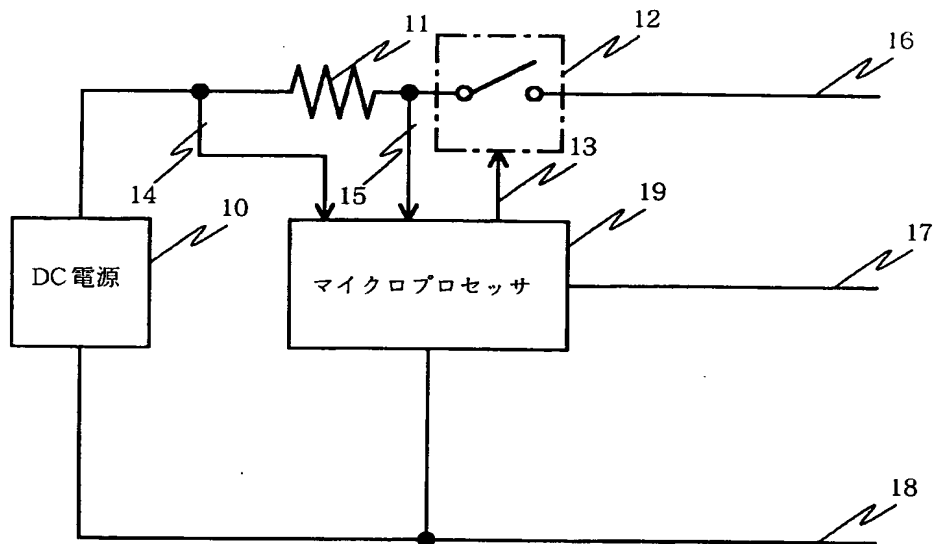
【図 1】



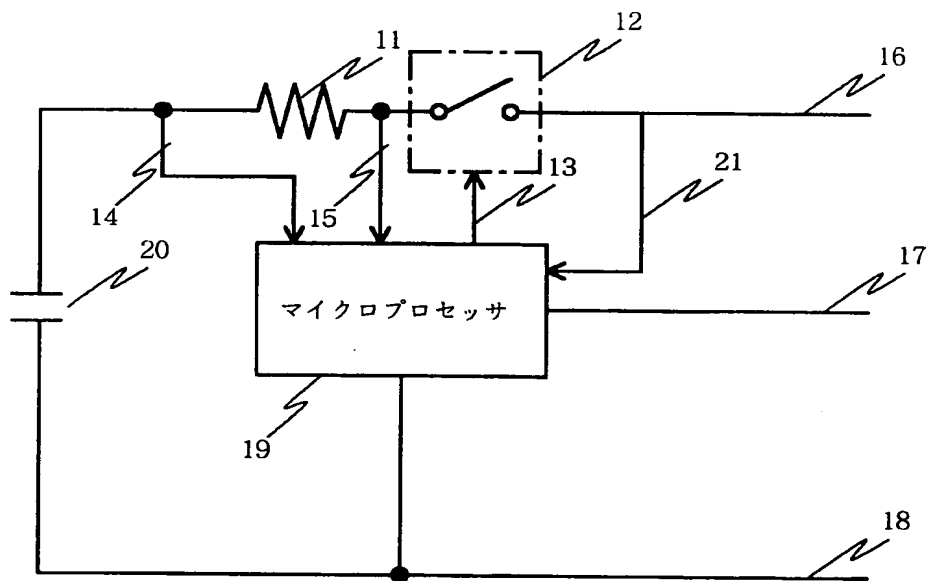
【図 6】



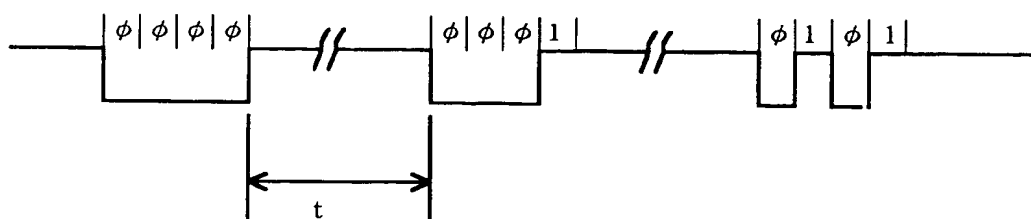
【図 2】



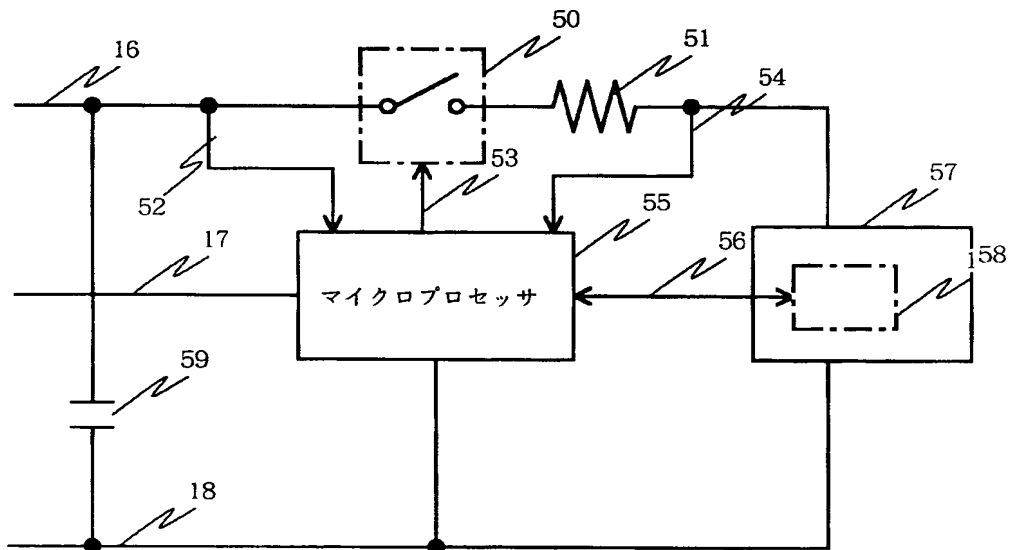
【図 3】



【図 7】

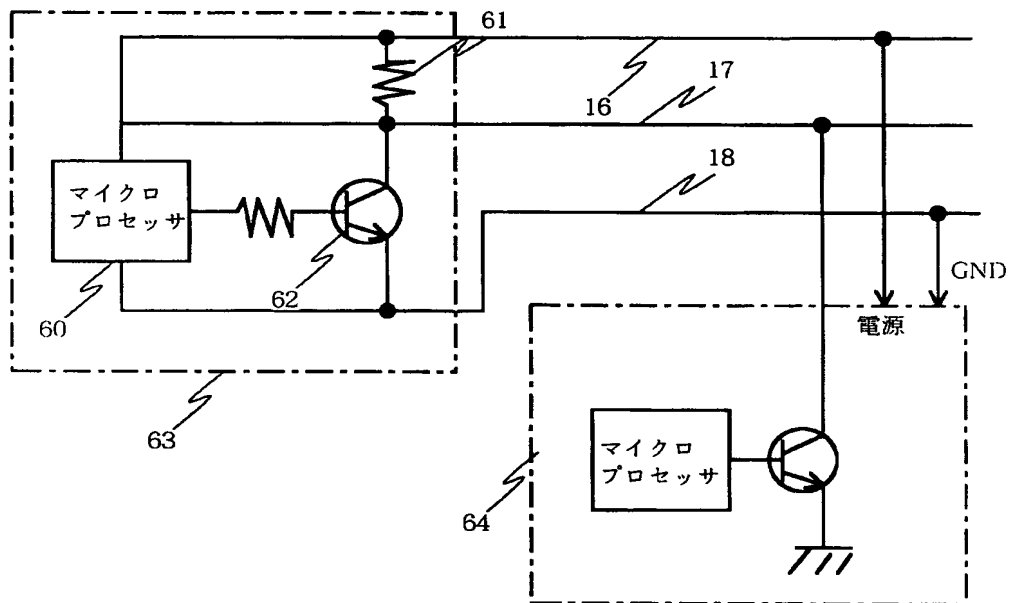


【図4】

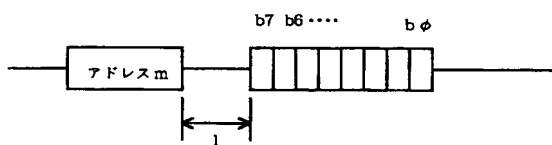


【図5】

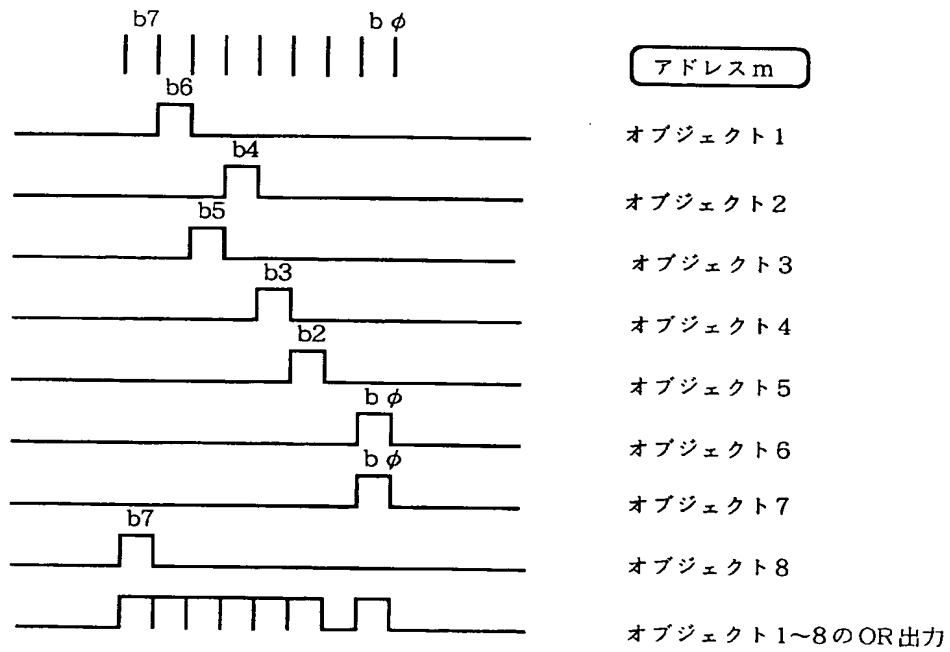
【図5】



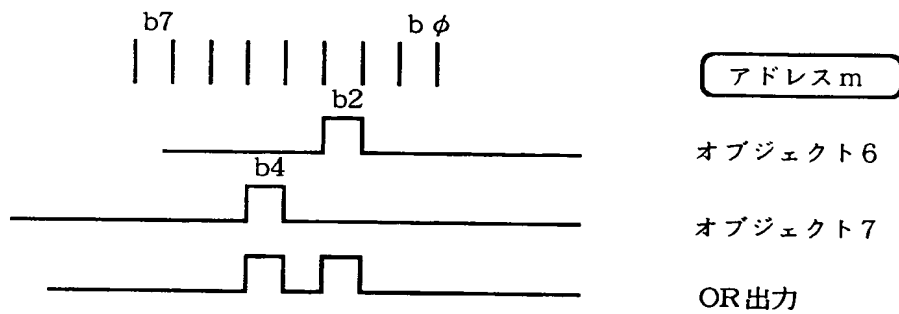
【図8】



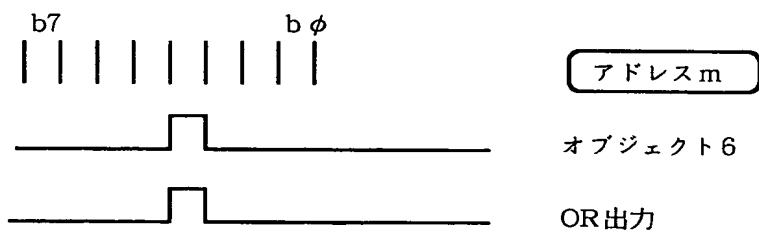
【図 9】



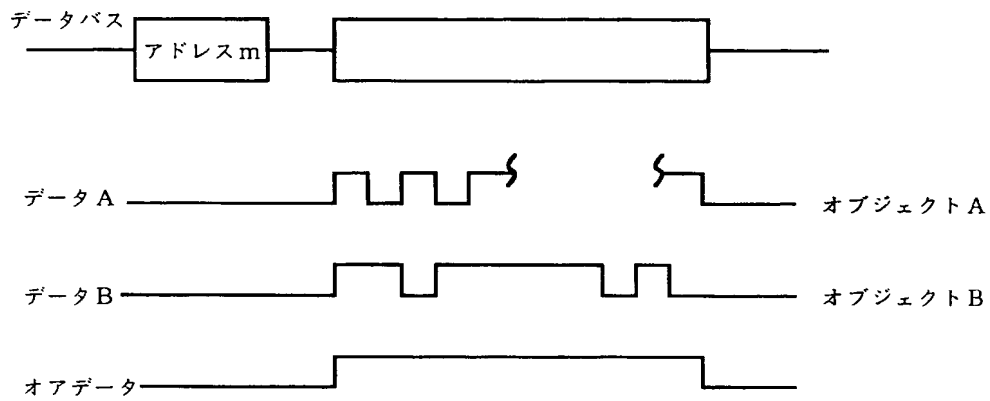
【図 10】



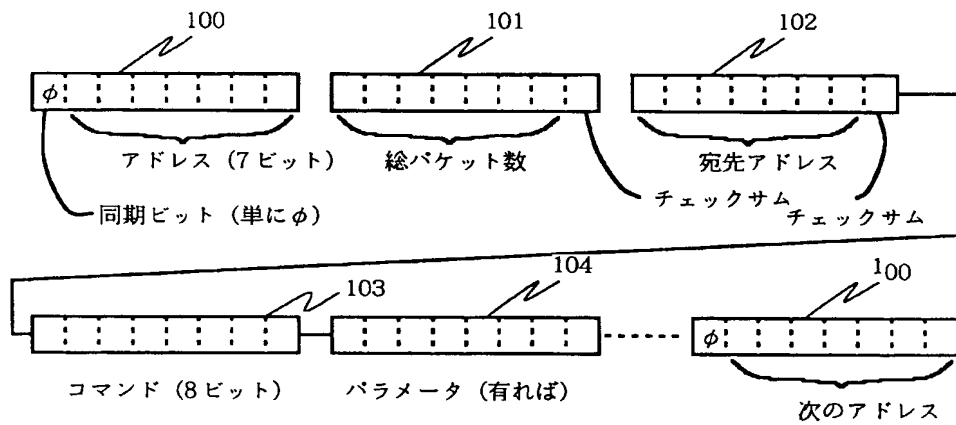
【図 11】



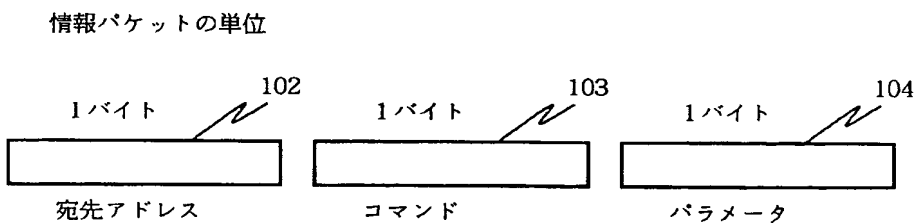
【図12】



【図14】

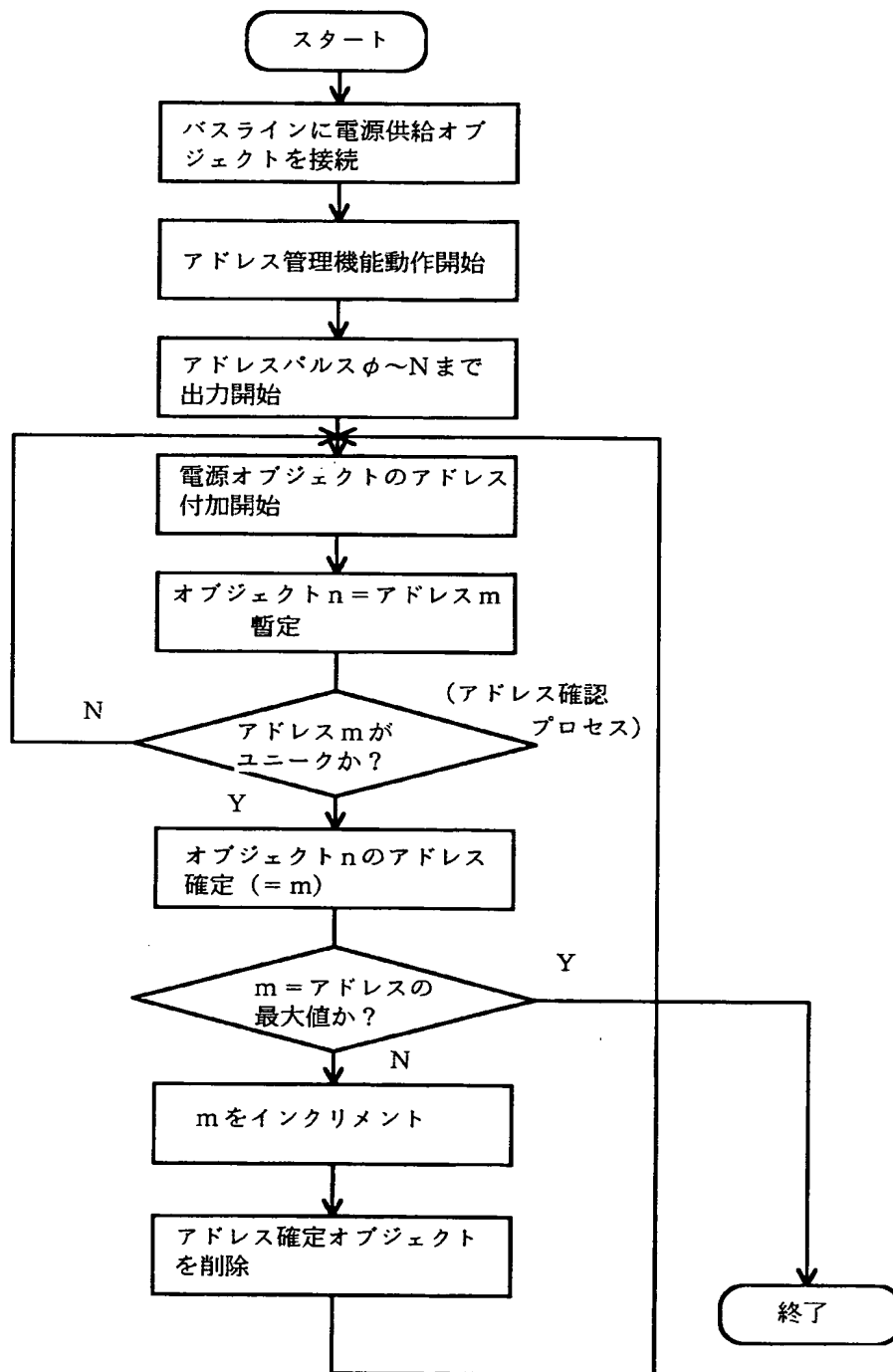


【図15】

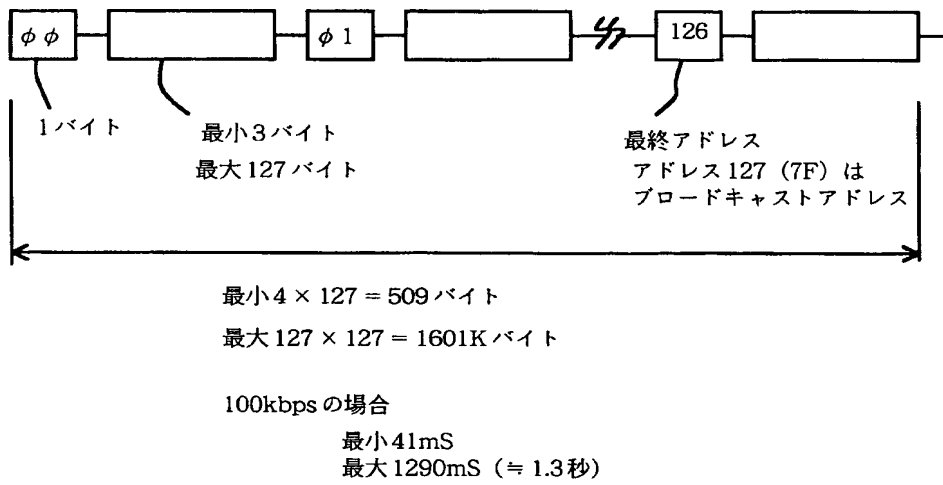




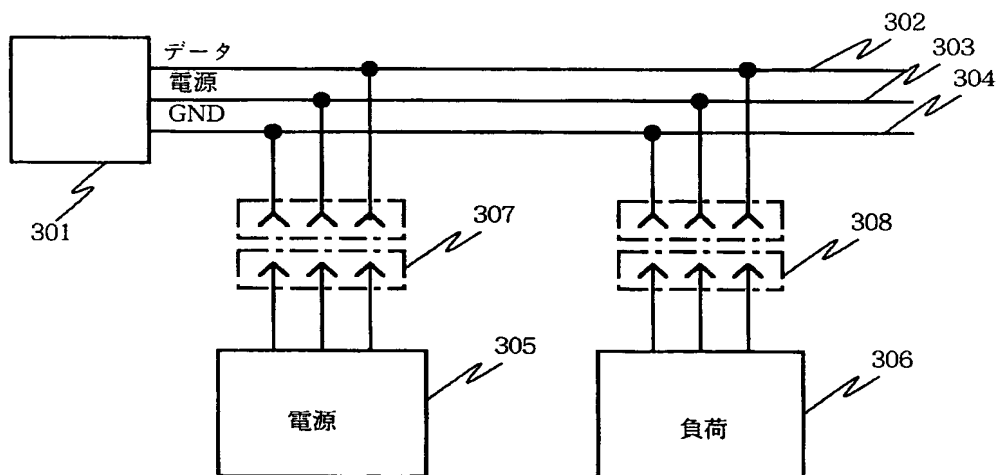
【図 13】



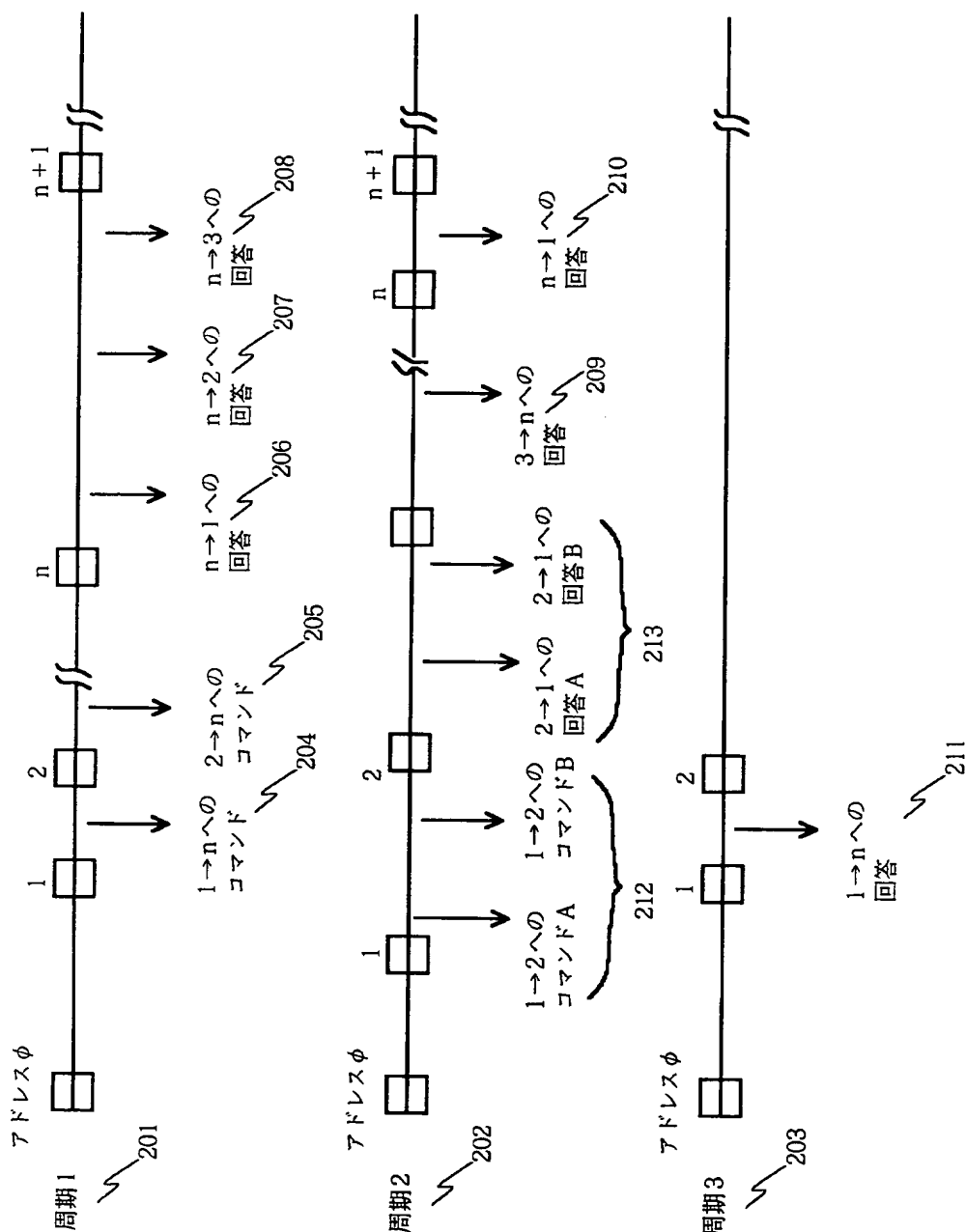
【図 16】



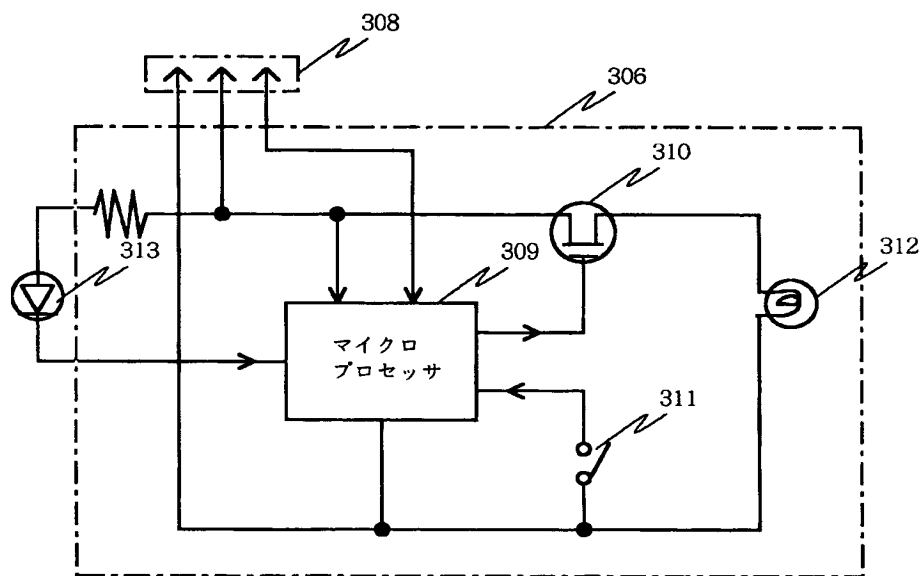
【図 18】



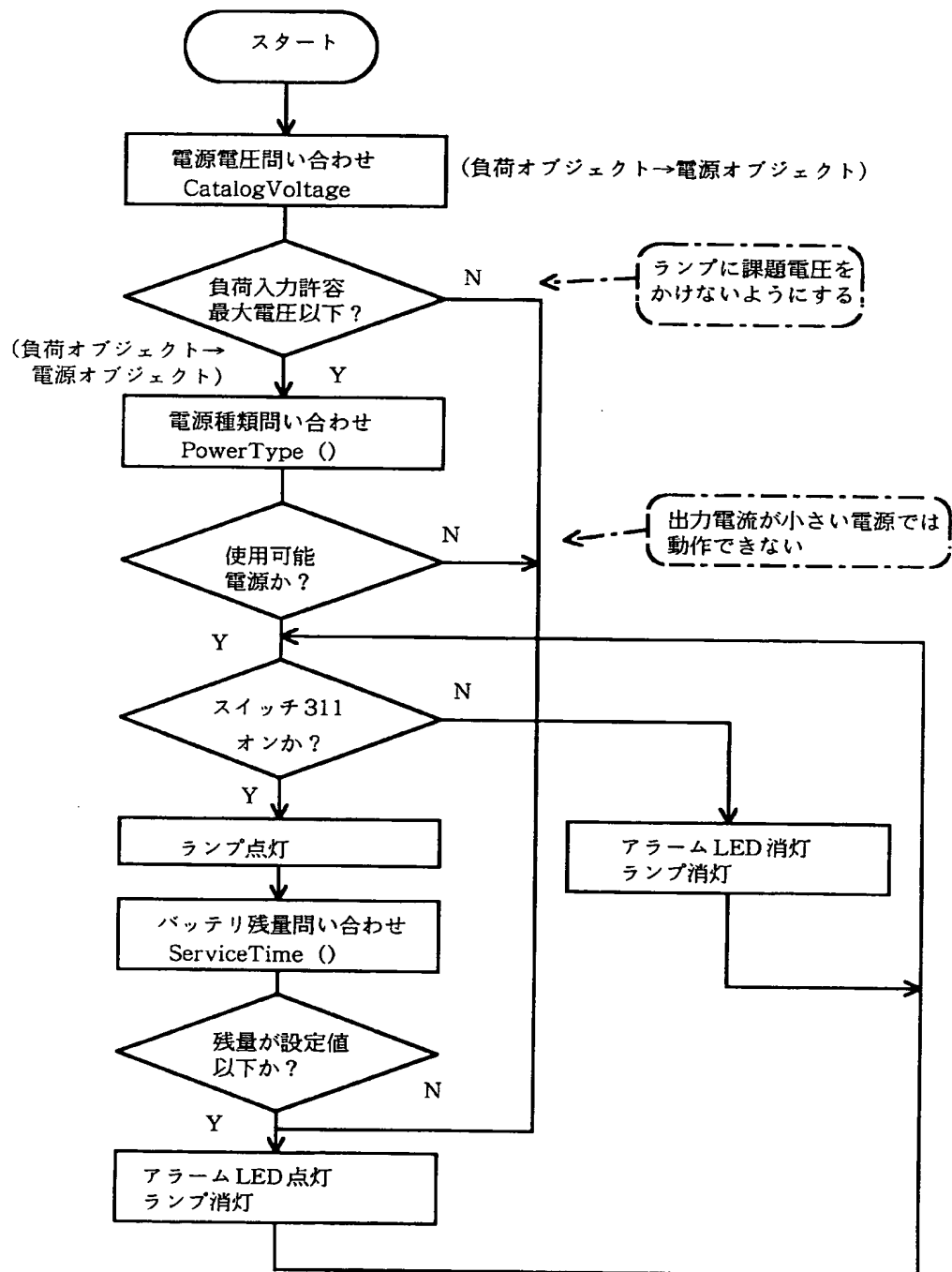
【図17】



【図 19】



【図20】



THIS PAGE BLANK (USPTO)